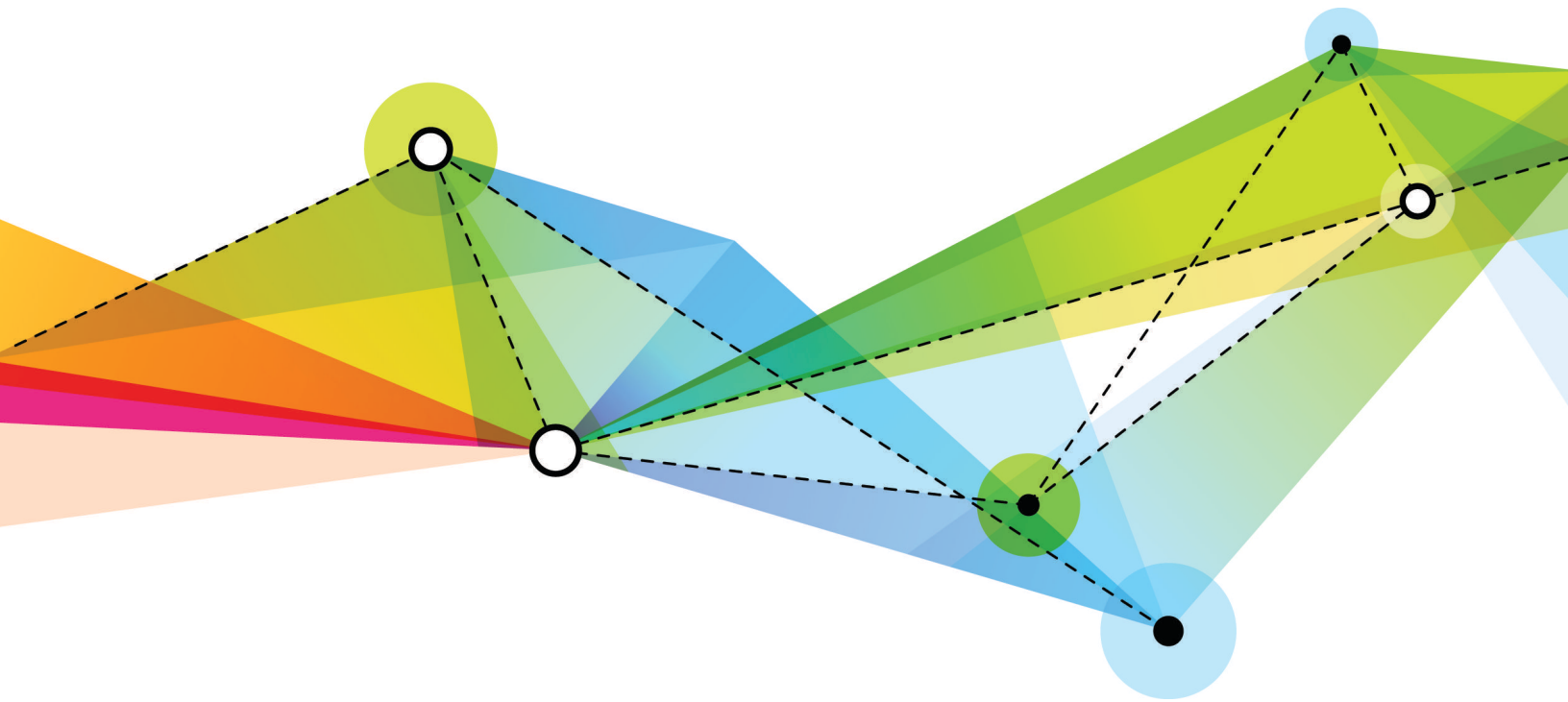




# Flexible Charge Importer



Edition: 1.1

Release date: November 30, 2016

Smile version: 6.0

Published by Inomial Pty Ltd

Suite 801, 620 Bourke St, Melbourne, Vic 3000, Australia

[www.inomial.com](http://www.inomial.com) • +61 3 9663 3554 • [sales@inomial.com](mailto:sales@inomial.com) • [support@inomial.com](mailto:support@inomial.com)

Copyright © 2016, Inomial Pty Ltd. Commercial in confidence.

# Flexible Charge Importer

---

## Introduction

---

The flexible charge importer provides a means for importing invoice line items into Smile from an external system and assigning them to particular accounts.

The flexible charge importer uses JExpr to express how Smile should handle an imported CSV column. For more information, see the *JExpr Language Guide*.

This document describes how to configure a Smile flexible importer to process a CSV charge import file.

**Note:** JExpr is intended for advanced users familiar with programming languages. [Contact Inomial](#) for assistance.

## Charge importers

---

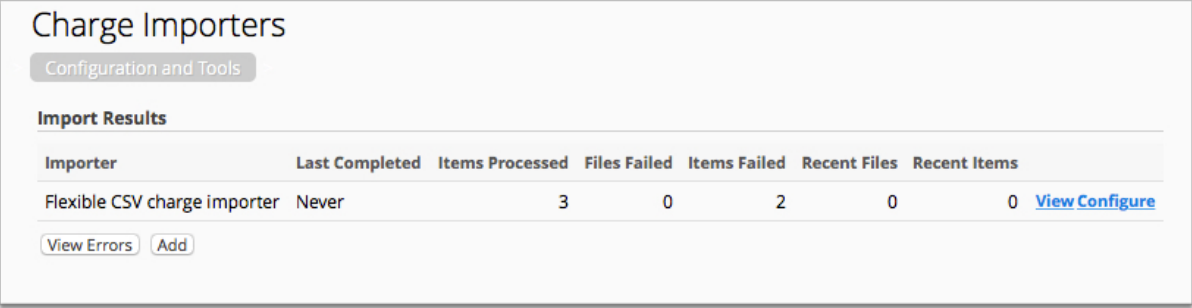
Charge importers are used for importing invoice line items into Smile, to be assigned to accounts.

A configured charge importer specifies the importer type, location or method of retrieving input charge files, configuration options applied during import, error actions, relevant services and the task schedule of the import action.

Importer types are internally defined in Smile. Importer types are advanced configuration. For more information, [contact Inomial](#).

The Charge importers main page displays a summary list of configured importers and import results. The following actions can be taken on configured importers:

- **View**—displays current import results for an importer, including the number of successful and failed file and item events.
- **Configure**—displays the configuration of the importer. The configuration of an importer can be edited.
- **View Errors**—displays import errors by importer, failure type and missing subscriptions. For more information about how to check for failed import items, see the *User Guide*.



The screenshot shows the 'Charge Importers' page in Smile. It features a 'Configuration and Tools' tab and an 'Import Results' section. A table displays the following data:

Importer	Last Completed	Items Processed	Files Failed	Items Failed	Recent Files	Recent Items	
Flexible CSV charge importer	Never	3	0	2	0	0	<a href="#">View Configure</a>

Below the table are two buttons: 'View Errors' and 'Add'.

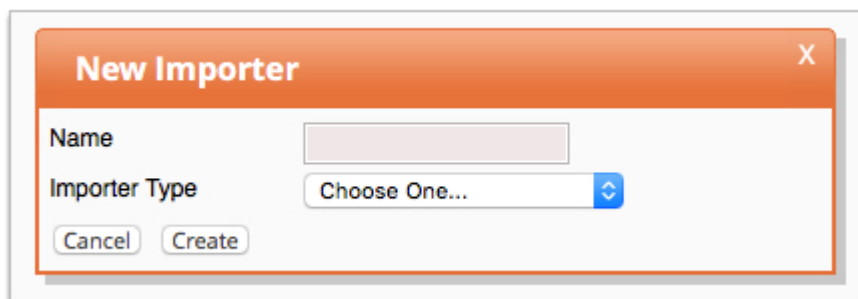
**Figure 1: Smile Charge Importers**

## Add an importer

Smile charge importers define how Smile acquires invoice line items. An appropriate **Importer Type** is required to add an importer.

This task explains how to add an importer.

1. Select **Charge importers** under **Accounts Receivable** on the Configuration and Tools page.  
The Charge Importers page is displayed.
2. Click **Add**.  
A **New Importer** window is displayed.



**Figure 2: New Importer window**

3. Type a name for the importer in the **Name** field.
4. Select **Flexible CSV Charge Importer** from the **Importer Type** drop-down.  
**Importer Types** are internally defined. For more information, [contact Inomial](#).
5. Click **Create**.  
The Configure importer page is displayed.
6. Complete the required importer configuration fields.  
Configuration options displayed are determined by the selected **Importer Type**.
7. Click **Manage Scheduled Task** to define when and how often the importer runs.  
For more information, see the *Configuration Guide*.
8. Click **Save**.  
The Smile Importers page is displayed. The importer is added to the **Import Results** summary list.

## Importer field properties

---

The following fields are available to configure non-JExpr properties of the flexible importer.

<b>Name</b>	Specifies a descriptive name of the importer.
<b>Spool subdirectory for fetched files</b>	Specifies the Smile subdirectory in which Smile places files to be processed.
<b>Remote Hostname</b>	Specifies the hostname of the remote server from which Smile retrieves the charges CSV.
<b>Remote port</b>	Specifies the port number of the remote server from which Smile retrieves the charges CSV. Default port number is 22.
<b>Remote username</b>	Specifies the username that Smile provides to the remote host.
<b>Filename of local private key (id_dsa)</b>	Specifies the name of the file containing the private key Smile users to authenticate the SFTP session.
<b>Remote directory</b>	Specifies the directory on the remote server from which Smile begins fetching files.
<b>Remote filename regex</b>	Specifies the regular expression defining which files to fetch. Importer will only download files matching this pattern. Leave blank to use the default.
<b>Traverse the remote directory recursively</b>	Specifies that the importer will traverse the remote directory recursively before proceeding.
<b>Field separator</b>	Specifies the field separator used in the CSV file being imported. Default separator is , (comma).
<b>Age threshold to be <i>n</i> hours "recent"</b>	Specifies the number of hours in the past of when files are still considered recent.
<b>Error Ticket</b>	<p>Specifies if a helpdesk ticket is created by Smile in the event of an import error.</p> <ul style="list-style-type: none"> <li>• <b>Don't raise tickets on errors</b></li> <li>• <b><i>template ticket name</i></b></li> </ul> <p>For more information on template tickets, see the <i>Configuration Guide</i>.</p>
<b>Manage Scheduled Task</b>	<p>Specifies when and how often the importer runs to automatically import charges.</p> <p>For more information on task scheduling, see the <i>Configuration Guide</i>.</p>

## JExpr field properties

---

The following fields are used to configure the JExpr properties of the flexible importer.

### Skip expression

**Values:** true, false

**Purpose:** Specifies header and footer rows of the CSV file. When true the entry is skipped.

**Example:**

```
$1 != "Charge"
```

Column 1 is not equal to "Charge". Any rows that do not contain the text "Charge" in the column 1 cell will be skipped.

### Item validity expression

**Values:** true, false

**Purpose:** Used to detect malformed rows. When false the entry is not valid

**Example:**

```
$$ == 4
```

There must be exactly 4 columns for an item row.

### Account/Subscription USN

**Format:** String

**Purpose:** Specifies the account or subscription that this charge will be recorded for.

**Example:**

```
$1
```

The account/subscription USN is the first column.

### Item code

**Format:** String

**Purpose:** Indicates what is being purchased. This should be the same code as shown in Invoice items menu in Configuration and Tools.

**Example:**

```
$2
```

The item code is the second column.

### Freeform text field to describe the item.

**Format:** String

**Purpose:** This is a freeform text field to describe the item. If unspecified, then the item description, as shown in Invoice items menu in Configuration and Tools, for this item is used.

**Example:**

```
$3
```

The item description is the third column.

**Start date**

**Format:** Java date (java.util.Date)

**Purpose:** Specifies the start date of item, inclusive. If unspecified, then the local system date at which each item is imported will be used.

**Example:**

```
parsers.parseDate("yyyy-MM-dd", $4)
```

The start date is given as the fourth column, in ISO 8601 format.

**End date**

**Format:** Java date (java.util.Date)

**Purpose:** Specifies the end date of item, inclusive. If unspecified, then the start date will be used.

**Example:**

```
parsers.parseDate("yyyy-MM-dd", $5)
```

The end date is given as the fifth column, in ISO 8601 format.

**Item quantity**

**Format:** BigDecimal

**Purpose:** Specifies item quantity. If unspecified, 1 will be presumed.

**Example:**

```
$6::BigDecimal
```

The item quantity is given as the sixth column.

**Number of instances**

**Format:** Integer

**Purpose:** Specifies item count. If unspecified, then 1 will be presumed.

**Example:**

```
$7::Integer
```

The item count is the seventh column.

**Total charge amount**

**Format:** BigDecimal

**Purpose:** Specifies the total charge amount, excluding tax. If omitted, will be computed as: quantity x item rate.

**Example:**

```
$8::BigDecimal
```

The total charge amount is the eighth column.

**Tax calculation mode**

**Format:** String

**Purpose:** Indicates how the tax component is to be calculated for this item. Should be one of the following options. If unspecified, then `Compute` is implied.

- **Compute**—Smile calculates tax automatically
- **Provide**—Use the already-provided value from the **Pre-calculated tax amount** expression
- **NA**—No tax is applicable for this item

**Example:**

```
$9
```

Tax calculation mode is derived from the ninth column.

**Pre-calculated tax amount**

**Format:** BigDecimal

**Purpose:** Specifies the pre-calculated tax amount from external system. Only required if **Tax calculation mode** is `Provide`.

**Example:**

```
$10::BigDecimal
```

Pre-calculated tax amount is obtained from the tenth column.

**Cost centre**

**Format:** String

**Purpose:** Specifies the name of cost centre. The cost centre must already exist on the subscription's account. Names are matched case-sensitively. If unspecified, the subscription's default cost centre is used.

**Example:**

```
$11
```

The cost centre is the eleventh column.

**Purchase order number**

**Format:** String

**Purpose:** Specifies the purchase order number to assign for this charge. If unspecified, the subscription's default purchase order is used.

**Example:**

\$12

The purchase order number is the twelfth column.

**Office name**

**Format:** String

**Purpose:** Specifies the name of office location where item was purchased. If unspecified, no office location is recorded for the line item.

**Example:**

\$13

The office name is the thirteenth column.

**Raising action**

**Format:** String

**Purpose:** Indicates how the item is raised. Should be one of the following options. If unspecified, then `Defer` is implied.

- **Defer**—Smile raises an item as a deferred charge on the account
- **Leave\_open**—Smile raises an item on a new open invoice
- **Close**—Smile raises an item on a new closed invoice

**Example:**

\$14

Raising action is derived from the fourteenth column.

**Invoice identifier**

**Format:** String

**Purpose:** Line items with the same invoice identifier are raised together in the same invoice. Only required if **Raising action** is `Leave_open` or `Close`.

**Example:**

\$14

The invoice identifier is the fourteenth column.



## JExpr dictionaries

---

The JExpr language can feature pre-defined dictionaries for certain parts of Smile that accept JExpr expressions for their configuration.

Dictionaries are a list of pre-defined functions. These functions can take zero or more arguments, each argument being a JExpr expression itself. The function will return an expression of a particular JExpr type (or may return null).

A function call in JExpr looks like the following:

```
parsers.parseDate( "yyyy-MM-dd" , $1 );
```

In this example:

- `parsers` is the dictionary namespace that contains the function
- `parseDate` is the name of the function
- This function takes two arguments (both of string type), and returns a Java date/timestamp (`java.util.Date`) value
- Parentheses encapsulate all of the arguments
- Commas separate individual arguments

## parsers dictionary

This dictionary contains functions that direct how specified strings should be parsed.

### parseDate ( )

```
parsers.parseDate(format, sourceText)
```

Parses a string containing a date and/or timestamp specification in a particular structured format.

#### Parameters

##### format

**Format:** string

**Purpose:** Describes the expected layout of the individual date/time components within the sourceText argument.

The format string can contain the following (case-sensitive) placeholders:

<b>yy</b>	Two-digit year. For example, 99 for 1999.
<b>yyyy</b>	Four-digit year. For example, 1999.
<b>MMMM</b>	Full month name. For example, July.
<b>MMM</b>	Three-letter month abbreviation. For example, Jul for July.
<b>MM</b>	Two-digit numeric month. For example, 07 for July.
<b>dd</b>	Two-digit day in month. For example, 23.
<b>HH</b>	Two-digit hour-in-day. For example, 00 thru 23 inclusive, for 24-hour time.
<b>hh</b>	Two-digit hour-in-day. For example, 01 thru 12 inclusive, for 12-hour time.
<b>mm</b>	Two-digit minute in hour. For example, 00 thru 59 inclusive.
<b>ss</b>	Two-digit second in minute. For example, 00 thru 59 inclusive.
<b>SSS</b>	Three-digit millisecond value. For example, 000 thru 999 inclusive.
<b>z</b>	Time zone. For example, UTC+10:00.
<b>Z</b>	RFC 822 time zone. For example, +1000 for 10 hours ahead of UTC.
<b>a</b>	AM/PM marker. For example, AM or PM.

To escape a letter (indicate that it needs to be matched verbatim), enclose it in single quotes ' '. To match a single quote directly, write two single quotes consecutively '' (no space between quotes).

**Note:** All the sequences supported by the Java Foundation Class `java.text.SimpleDateFormat` are permitted for this parameter. For more information, see <http://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html>

The letters 'G', 'Y', 'L', 'W', 'w', 'D', 'F', 'E', 'u', 'k', 'K', 'X' are also supported as per the above Java specification.

Any other characters, such as punctuation, spaces or letters not mentioned above, will be matched verbatim.

### sourceText

**Format:** string

**Purpose:** Specifies the date string to be parsed.

### Example: A parseDate request

The following example is a `parseDate` request containing a string for the `format` argument and `sourceText` of the date to parse.

```
parsers.parseDate("dd/MM/yyyy HH:mm:ss", "31/12/1999 23:59:59")
```

### Result

Returns a `java.util.Date` instance containing the parsed date/timestamp.

### Example: A returned parseDate request

This example shows the returned `parseDate` request.

```
31st December 1999 at 11:59:59pm
```

## importerContext dictionary

This dictionary contains functions that return information about the current input file.

### getFilename()

```
importerContext.getFilename()
```

Returns the file name of the file currently being processed.

### Parameters

No parameters.

### Returns

Returns a string value.